DRAFT Oct 2006

CETerm RFID Guide for Symbol Mobile Computers

for Version 5.1 or later

Naurtech Terminal Emulation and Web Browser Smart Clients

for Windows CE Devices

CETerm | CE3270 | CE5250 | CEVT220

Copyright Notice

This document may not be reproduced in full, in part or in any form, without prior written permission of Naurtech Corporation.

Naurtech Corporation makes no warranties with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Naurtech Corporation, reserves the right to revise this publication and referenced software without any obligation to notify any person or organization of such revision or changes.

Trademarks

CETerm®, CE3270™, CE5250™, CEVT220™ are trademarks of Naurtech Corporation.

Other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Software Version

This document is for Version 5.1 or later of Naurtech Smart Clients.

Table of Contents

Copyright Notice 2
Trademarks 2
Software Version 2
Table of Contents 3
Preface 4
Assumptions 4
Conventions used in this Manual 4
Additional Documentation 5
Online Knowledgebase 5
1.0 Introduction 6
  1.1 Feature Highlights 7
2.0 Getting Started 8
  2.1 RFIDInventory Scripts 8
  2.2 RFIDInventory HTML Page 9
  2.3 Running the RFID Inventory Sample 9
3.0 CETerm RFID Automation Objects 10
  3.1 The CETerm Object 10
    Properties 10
  3.2 The RFIDReader Object 10
    Methods 11
    Properties 14
  3.3 The RFIDTag Object 18
    Methods 18
    Properties 18
4.0 CETerm RFID Events 19
  4.1 The OnRFIDInventoryDone Event 19
    Syntax 19
    Example 19
  4.2 The OnRFIDTrigger Event 20
    Syntax 20
    Example 20
  4.3 The OnRFIDReadNewTag Event 20
    Syntax 21
    Example 21
  4.4 The OnRFIDFilterMatch Event 21
    Syntax 21
    Example 21
Appendix 1 – RFID Constants 22
  General Constants 22
  Gen2 Constants 22
  ReadCap and WriteCap Constants 24
  Trigger Constants 25
  Utility Functions 25
Appendix 2 – RFID Inventory Sample Scripts 27
Appendix 3 – RFID Inventory Sample HTML 31
Index 34

Preface

All of us at Naurtech Corporation constantly strive to deliver the highest quality products and services to our customers. We are always looking for ways to improve our solutions. If you have comments or suggestions, please direct these to:

Naurtech Corporation
e-mail: contact@naurtech.com
Phone: +1 (425) 837.0800

Assumptions

This manual assumes you have a working knowledge of:

- Microsoft Windows user interface metaphor and terminology.
- Stylus based touch screen navigation terminology.
- Basic programming and scripting concepts.
- Dynamic HTML, the browser DOM, and JavaScript.
- Basic operations and requirements of the host applications you want to access with the Naurtech smart client.

Conventions used in this Manual

This manual uses the following typographical conventions:

- All user actions and interactions with the application are in bold, as in **`[Session][Configure]`**

- Any precautionary notes or tips are presented as follows

  Tip: Text associated with a specific tip

- ☞ represents new version specific information
- All text associated with samples is presented as follows.

```
/*alert*/
OS.Alert("Script done.");
```

Additional Documentation

Naurtech RFID features are integral to Naurtech terminal emulation Smart Clients. Please refer to the User's Manual for detailed installation and configuration information. Also refer to the Naurtech Scripting Guide for details on scripting features within CETerm. These manuals may be downloaded from the "Support" section of our website.

Online Knowledgebase

Although we continually strive to keep this manual up to date, you may find our online support knowledgebase useful for the latest issues, troubleshooting tips and bug fixes. You can access the support knowledgebase from our website at:

www.naurtech.com → Support → Knowledgebase

1.0 Introduction

The Naurtech CETerm Smart Clients provide a robust and flexible environment for Terminal Emulation (TE) and Web based applications on a mobile device. This document describes the RFID features available within CETerm for the Symbol family of mobile computers, which contain RFID readers.

CETerm fully integrates RFID support using the CETerm Scripting Engine to make the RFID reader available to both TE and web browser sessions. All native capabilities of the Symbol RFID reader are accessible within the CETerm Scripting Engine. CETerm uses the industry standard JavaScript programming language which is familiar to most programmers and is a driving force behind the rich next-generation web based applications. This new class of web applications is sometimes referred to as “Web 2.0” using Asynchronous JavaScript and XML (AJAX). CETerm brings this mature and rich language to the TE user to provide more productive TE applications. Scripting can also interact with web browser sessions to enrich and extend existing web applications on the mobile device.

Together, JavaScript and the CETerm RFID Control Object allow nearly limitless flexibility for reading, writing, inventory, and other RFID tag operations. The browser provides an especially powerful environment to build highly interactive stand-alone or casually-connected RFID applications. Legacy applications using terminal emulation can also be “RFID enabled” with little or no changes to the existing application.

This guide describes how to use the CETerm RFID Control Object within the CETerm Scripting Engine. The properties, methods and events of the RFID control are documented together with some simple examples. For full documentation of the CETerm Scripting Engine, the steps for writing and running scripts and the features provided through the CETerm Automation Objects, refer to the CETerm Scripting Guide. Please consult the standard references for details on JavaScript (or JScript) syntax and XML. You may also need to consult standard references for HTML syntax, the browser Document Object Model (DOM), and other aspects of Dynamic HTML if you are scripting web browser features. Additional information on the CETerm Web browser features can be found in the CETerm Web Browser Programming Guide. Please refer to the Naurtech User’s Manual for details on basic usage and configuration of the Naurtech clients

We believe that the CETerm RFID and Scripting features will enrich and extend the capabilities of your TE and browser applications. Explore a little deeper and we think you will be amazed at the possibilities for building powerful business applications.

1.1 FEATURE HIGHLIGHTS

Following are some of the special features in CETerm RFID

- **JavaScript.** Naurtech uses the industry standard JavaScript scripting language. This powerful language is familiar to programmers and non-programmers world-wide as the core of rich web applications. With JScript, the Microsoft version of JavaScript, additional features are available such as the ability to use ActiveX objects in scripts.

- **Full Support for Symbol RFID features.** The CETerm RFID Control Object provides access to all native features of the Symbol RFID device. This includes setting all parameters, controlling reads, writes, locks and inventories, and retrieving statistical reports.

- **Full Access from both TE and Browser Sessions.** Both terminal emulation and browser sessions can control the RFID device. The user can switch between multiple sessions, with each having different settings and operations. For example, a TE session may be performing inventory operations whereas a web session may be reading barcodes and programming RFID tags with the corresponding information.

- **CETerm Automation Objects.** CETerm Automation Objects are available to access and control the state of CETerm, the state of a TE or web browser session, the mobile device, and the Windows CE Operating System. Together these objects provide a rich set of features to build complex RFID operations, including stand-alone and casually-connected applications which store RFID results locally and later forward to a host system.

2.0 Getting Started

This section describes some common ways that RFID features can be used within CETerm. Here we describe a web browser sample application that receives RFID inventory results. This sample application demonstrates how RFID data can be passed to a web browser application and how some basic RFID configurations can be set from a browser page. For discussion, the application can be conveniently divided into a "scripts" component and a "HTML" component (page). Please note that the HTML page will also have JavaScript which runs separately in the browser JavaScript Engine.

2.1 RFIDINVENTORY SCRIPTS

The RFID Scripts for this sample are stored on the device and are loaded into the CETerm JavaScript engine at program startup. It is also possible to download scripts dynamically from a web server or TE server. The main scripts are contained in the RFIDInventory.js file and additional RFID utility scripts are in the RFIDUtils.js file

To prepare for the sample program, both RFIDInventory.js and RFIDUtils.js must be imported into CETerm using the configuration dialog under:

`[Session]->[Configure]->[Options]->[Configure Scripting]`

On the **Scripts** tab select an empty script slot and tap **Edit** then **Import**. Select the file and check the **Load at Startup** option then tap **OK**.

Import both scripts as described above then on the **General** tab make sure that **Enable** is checked then tap **Re-Initialize** and **OK** to close the dialogs. See the CETerm Scripting Guide for more details on loading scripts and configuring the scripting engine.

The RFIDInventory scripts are shown in Appendix 2. The function `RFIDInitInventory` must be run first, and this is invoked by the HTML page when it loads. After initialization, the important functions are the event handlers `OnRFIDTrigger` and `OnRFIDInventoryDone`. When the trigger is pulled, `OnRFIDTrigger` starts the inventory and stops it when the trigger is released. During the inventory cycles, `OnRFIDInventoryDone` is called when there are results to be displayed.

`OnRFIDInventoryDone` formats the results for the web browser and passes them to the browser with a similarly named `OnRFIDInventoryDone` method in the JavaScript of the web page. Please note that the web browser JavaScript cannot directly access the RFID Control Object because it belongs to the

CETerm JavaScript engine. This does not prevent the web page from requesting or receiving RFID results. Rather it simply requires that some scripts run in the CETerm environment.

Another notable function is `RFIDUpdateBrowserParams`. This function is run to update the current RFID inventory configuration based on the current settings within the HTML page.

The RFIDInventory scripts also have the ability to direct RFID data to a TE session if that is the currently active session.

2.2 RFIDINVENTORY HTML PAGE

The inventory sample HTML page is contained in RFIDInventory.htm and is shown in Appendix 3. The HTML file should be copied to the device and placed in the "\" root directory.

Select a session in CETerm for the sample and check the **HTML** Host Type. Configure the Host Address as:

```
file:///RFIDInventory.htm
```

The sample page contains three general sections. These are the top "instruction section", the middle "results section" and the bottom "configuration section".

The instruction section provides feedback to the user for operating the sample application. The results section shows the results of tag inventories, and the bottom section can be used to change the desired tag classes and good-read feedback.

2.3 RUNNING THE RFID INVENTORY SAMPLE

After the above configuration, to run the RFID Inventory sample simply connect the corresponding session with

`[Session]->[Connect]`

Pulling the trigger will now start a tag inventory. Results will be displayed when the trigger is released. You can change the desired tag classes by unchecking or checking the appropriate boxes.

3.0 CETerm RFID Automation Objects

This section describes the RFID automation objects available from the CETerm script engine. The `RFIDReader` object provides control and configuration of the RFID device. The `RFIDTag` object provides access to the properties of a tag that has been read.

The automation objects are accessed in a hierarchical manner similar to the Document Object Model (DOM) of a webpage. The two top-level objects are `CETerm` and `OS`. The `CETerm` object provides access to application specific features whereas the `OS` object provides access to generic Operating System (OS) features such as files and the device registry.

The `CETerm` object provides access to the `RFIDReader` object. Below, we describe just a small portion of the `CETerm` object. For full documentation of the CETerm object refer to the CETerm Scripting Guide.

3.1 THE CETERM OBJECT

The `CETerm` object gives access to CETerm features, settings, and session state. The `RFIDReader` object is accessed as a property of the CETerm object. Here we describe only the properties and omit all method descriptions.

Properties

The `CETerm` object has several application level properties.

Property	Description	Values
ActiveSession	Current active session. (read only)	1-4
MaxSession	Maximum session index. (read only)	4
Message	Returns message object. (read only)	object
RFIDReader	Returns RFID object. (read only)	object
TextInput	Return text input object. (read only)	object

3.2 THE RFIDREADER OBJECT

The `RFIDReader` object provides configuration and control of the RFID device. Although typically used to read and inventory tags, the `RFIDReader` also provides access to program, lock and kill operations. This section documents the methods and properties of the `RFIDReader` object.

Methods

The following methods are available

Method	Action
EnableReader	Enable or disable the reader.
EnableTrigger	Enable or disable the trigger
GetCommandStatusText	Returns text description of Symbol error status
GetCapList	Returns a list of Symbol RFID capabilities
GetCapInfo	Return an XML description of a capability
SetCapValue	Set a single BYTE, WORD, or DWORD capability.
GetCapValue	Get a single BYTE, WORD, or DWORD capability.
GetStats	Return an XML description of the current statistics.
StartTagInventory	Start a tag inventory cycle.
ReadTag	Read a single tag.
ProgramTags	Program tags within range.
EraseTag	Erase tags within range.
LockTag	Lock tags within range.
KillTag	Kill tags within range.
SetScrollInventoryDefaults	Set to default “scroll” inventory parameters.
SetTreeInventoryDefaults	Set to default “tree” inventory parameters.
TagFilterAdd	Add a tag filter.
TagFilterRemove	Remove a single tag filter.
TagFilterRemoveAll	Remove all tag filters.
GetTag	Returns the requested tag object.
GetTagID	Returns the tag ID of the specified tag.
ClearInventory	Clear the current tag inventory.

status = EnableReader (enabled)

Enable the reader if **enabled** is true, disable if **enabled** is false. Returns zero on success or non-zero if fails.

status = EnableTrigger (enabled)

Enable the trigger if **enabled** is true, disable if **enabled** is false. Returns zero on success or non-zero if fails.

text = GetCommandStatusText (status)

Return the text description of a Symbol error status.

text = GetCapList ( )

Return the text list of Symbol RFID Reader capabilities.

capXML = GetCapInfo (capID)

Return an XML description of the capability specified by **capID**.

status = SetCapValue (capID, value)

Set a capability that uses a single BYTE, WORD, or DWORD value. Returns zero on success or non-zero if fails.

value = GetCapValue (capID)

Returns the value of a capability that uses a single BYTE, WORD, or DWORD, value. The property **LastGetCapStatus** will contain the status of the last call and can be used to check for errors.

statsXML = GetStats ( )

Return an XML description of the current reader statistics.

Tag Operation Methods

status = StartTagInventory ( )

Start the tag inventory action. Returns zero on success or non-zero if fails.

status = ReadTag ( )

Read a single tag. Returns zero on success or non-zero if fails.

status = ProgramTags (tagID, verifyCount, eraseAttempts, programAttempts)

Program all tags within range. Returns zero on success or non-zero if fails.

status = EraseTag (verifyCount, eraseAttempts)

Erase all tags within range. Returns zero on success or non-zero if fails.

status = SetScrollInventoryDefaults ( )

Set inventory parameters to the “scroll” inventory default values. Returns zero on success or non-zero if fails.

status = SetTreeInventoryDefaults ( )

Set inventory parameters to the “tree” inventory default values. Returns zero on success or non-zero if fails.

status = TagFilterAdd (index, bitMask, options)

Add a tag filter. Returns zero on success or non-zero if fails. See Section x for details on setting a **bitMask** and Appendix 1 for **options** values.

status = TagFilterRemove (index, bitMask, options)

Remove a tag filter. Returns zero on success or non-zero if fails. See Section x for details on setting a **bitMask** and Appendix 1 for **options** values.

status = TagFilterRemoveAll ( )

Remove all tag filters. Returns zero on success or non-zero if fails.

Inventory Access Methods

tagObject = GetTag(index)

Returns an object representing the specified tag in the current tag inventory. If there are no tags available or **index** is out of range returns a **null** reference. See the property **TotalTagCount** to find the number of tags available. The variable **index** is zero-based and has a valid range of 0 to **TotalTagCount-1**. See the `RFIDTag` object for a description of methods and properties available on the tag object.

tagID = GetTagID(index)

Returns the tag ID of the specified tag in the current tag inventory. If there are no tags available or **index** is out of range returns an empty value. See the property **TotalTagCount** to find the number of tags available. The variable **index** is zero-based and has a valid range of 0 to **TotalTagCount-1**.

status = ClearInventory ( )

Clears the current tag inventory. Returns zero on success or non-zero if fails.

Properties

The `RFIDReader` object has many properties to control the inventory and other operations. The properties are separated into several groups for convenience.

General Reader Properties

Property	Description	Values
ReaderEnabled	Enable state of reader (read only). Use the EnableReader method to change this state.	true, false
TriggerEnabled	Enable state of trigger (read only). Use the EnableTrigger method to change this state.	true, false
UseExtendedTagFormat	True to use the Symbol extended (EX) tag format for tag masks and inventory results.	true, false
ReaderNumber	Unique number identifying this reader.	0 – 255
ReaderPowerOn	Enable or disable power to the reader. See the Symbol RFID SDK for usage details.	true, false
RFPowerOn	Enable or disable RF power to the reader. See the Symbol RFID SDK for usage details.	true, false
RFWriteAttenuation	RF write attenuation level	0 – 255
RFReadAttenuation	RF read attenuation level	0 – 255
ReadDutyCycleOnTime	Reader on-time in millisec.	0 – 255
ReadDutyCycleOffTimePercent	Reader off-time in percent.	0 – 100
LastGetCapStatus	Status of last GetCap call. See Appendix 1 for status values (read only).	status
LastSetCapStatus	Status of last SetCap call. See Appendix 1 for status values (read only).	status
LastGetParamStatus	Status of last GetParam call. See Appendix 1 for status values (read only).	status
LastSetParamStatus	Status of last SetParam call. See Appendix 1 for status values (read only).	status

General Tag Properties

Property	Description	Values
TagMask	Tag mask for all actions.	bitMask
TagClass0Enabled	Enable Class 0 tags.	true, false
TagClass0PlusEnabled	Enable Class 0 Plus tags.	true, false
TagClass0ZumaEnabled	Enable Class 0 Zuma tags.	true, false
TagClass1Enabled	Enable Class 1 tags.	true, false
TagClassGen2Enabled	Enable Gen2 tags.	true, false
TagClassOtherEnabled	Enable other classes of tags.	true, false
ActionTagType	Tag type for action. See Appendix 1 for tagType values	tagType
SingulationFieldClass0	Singulation ID for write. See Appendix 1 for tagSingID values.	tagSingID
LockCode	Lock code for tag.	0 – 0xffffff

Generation 2 Tag Properties
See the Symbol RFID SDK for details on these properties

Property	Description	Values
Gen2SelectedFlagMode	Flag mode for operations.	ignore, set, clear
Gen2CommunicationSession	Session for read operations.	S0, S1, S2, S3
Gen2RequiredInventoriedFlag	Required flag for inventory.	A, B
Gen2InitialInventoryQ	Initial slots for inventory.	0 – 15
Gen2SelectMemoryBank	Memory bank for select operation.	EPC, TID, USER
Gen2GeneralMemoryBank	Memory bank for general operations.	EPC, TID, USER
Gen2StartBitAddress	Starting bit address for operation.	0 – 16000
Gen2WriteOptions	Write Options. See Appendix 1 for writeOption values	writeOptions
Gen2LockOptions	Lock Options. See Appendix 1 for lockOption values	lockOptions
Gen2LockMaskMode	Lock Mask Mode. See Appendix 1 for maskMode values	maskMode
Gen2LockAction	Lock Mask Mode. See Appendix 1 for lockAction values	lockAction
Gen2AccessPassword	Access password. 32-bit value	0–0xffffffff
Gen2KillPassword	Kill password. 32-bit value	0–0xffffffff

Inventory Acquisition Properties

See the Symbol RFID SDK for details on these properties. It is usually best to not alter the loop properties and simple set the "tree" or "scroll" inventory defaults using the methods `SetScrollInventoryDefaults` or `SetTreeInventoryDefaults`.

Property	Description	Values
InventoryType	Type of inventory search.	0 – tree, 1 – scroll
InventoryClearBeforeRead	Clear inventory before read.	true, false
InventoryWakeLoops	Overall tag wake loops.	1 – 25
InventoryReadLoops	Number of read loops within each wake loop.	1 – 25
InventoryWakesPerLoop	Number of wake actions in outer loop.	0 – 25
InventoryWakesAfterLoops	Number of wake actions after outer loops.	0 – 25
InventoryFrequencyHopping	Hop frequencies during read.	true, false
InventorySleepsAfterRead	Sleep interval after each tag read.	0 – 25
NewTagSoundEnabled	Generate a sound for each new tag read.	true, false
NewTagSound	Specifies the new tag sound.	text, wav file
NewTagScriptEnabled	Run the "OnRFIDNewTag" script for each new tag. WARNING: This script must be brief.	true, false
TagFilterDropSound	Generate a sound when a tag is dropped by a drop filter.	text, wav file
TagFilterPassSound	Generate a sound when a tag is passed by a pass filter	text, wav file

Inventory Result Properties

Property	Description	Values
TotalTagCount	Valid tags in inventory (read only).	0 – 200
TotalReadCount	Total reads for all tags (read only).	0 – max
NewTagCount	New tags read in last inventory cycle (read only).	0 – 200
ElapsedReadTime	Total elapsed time of read (read only) (millisec).	0 – max
LastSequenceNumber	Sequence number of last read (read only).	0 – max
MaximumTagCount	Maximum tags managed by reader.	200
InventoryLastStatus	Status of last Inventory cycle. See Appendix 1 for status values (read only).	status

3.3 THE RFIDTAG OBJECT

The `RFIDTag` object provides access to the properties of a tag after an inventory operation.

Methods

The `RFIDTag` object has no methods.

Properties

The `RFIDTag` object contains only read-only properties.

Property	Description	Values
Status	Returns status of last tag read. See Appendix 1 for status values.	status
AntennaNumber	Antenna number used for read.	0 – 10
DataLength	Number of bytes in tag	0 – max
FirstReadTime	Time of first tag read.	text
LastReadTime	Time of last tag read.	text
ReadCount	Count of reads of this tag.	0 – max
Type	Type of tag. See Appendix 1 for tagType values.	tagType
Format	Format of tag. See Appendix 1 for tagFormat values.	tagFormat
MemoryPageNumber	Page number. Available only in EX tag mode.	0 – max
MemoryPageOffset	Page offset. Available only in EX tag mode.	0 – max
MemoryPageLength	Page length. Available only in EX tag mode.	0 – max
ID	Tag ID value. Returned as a sequence of hexadecimal values separated by spaces.	text

4.0 CETerm RFID Events

This section describes the script events within the CETerm script engine that are triggered by the RFID reader. These events provide ways to handle various conditions in CETerm. The event handlers are arbitrary scripts.

The event model in CETerm uses specific event handler names to bind events to handlers. If the event handler function (e.g., `OnRFIDInventoryDone`) is defined in the script engine, it will be executed when the event occurs. There is no special command required to register or bind the function to the event. Event handlers can be re-defined at any time. If the handler is no longer needed, the function can be re-defined as empty.

Event	Fired when…
OnRFIDInventoryDone	An RFID inventory cycle completes.
OnRFIDTrigger	The trigger is pulled or released.
OnRFIDReadNewTag	A previously un-read tag is read.
OnRFIDFilterMatch	A tag is read which matches a drop or pass filter.

4.1 THE ONRFIDINVENTORYDONE EVENT

The `OnRFIDInventoryDone` event is fired when an inventory cycle completes. An inventory cycle is started with the `StartTagInventory` method. The duration of a cycle depends on the inventory parameters. Typically, if the trigger is held, a new cycle will begin immediately following the current cycle. When the trigger is released, the results can be processed and sent to a TE or browser session.

Syntax

```
function OnRFIDInventoryDone ( status, totalTags, newTags )
```

status – status of inventory read cycle
totalTags – total tags in inventory (since last cleared)
newTags – new tags read during this cycle

Example

```
See the RFIDInventory.js scripts for a sample use of
OnRFIDInventoryDone.
```

4.2 THE ONRFIDTRIGGER EVENT

The `OnRFIDTrigger` event is fired when the trigger is enabled and the trigger is pulled or released.

Syntax

```
function OnRFIDTrigger ( status )
```

status – trigger status flags.

Example

This example is from the RFIDInventory sample.

```
/* OnRFIDTrigger */
function OnRFIDTrigger( status )
{
    if (RFIDTriggerAllowed &&
        (status & TRIGGER_ALL_MASK))
    {
        // Trigger pulled
        RFIDTriggerAllowed = false;
        RFIDInventoryActive = true;

        Reader.ReaderPowerOn = true;
        Reader.StartTagInventory();
    }
    else
    {
        // If trigger not enabled or released
        // Stop inventory
        RFIDInventoryActive = false;

        // Usually PowerOn = false will stop inventory.
        // Sometimes if the trigger is released too quickly
        // it fails to stop the inventory.
        // To recover, pull and release the trigger again.
        Reader.ReaderPowerOn = false;
    }
}
```

4.3 THE ONRFIDREADNEWTAG EVENT

The `OnRFIDReadNewTag` event is fired when a new tag is read during an inventory. The event handler receives a handle to a tag object containing the data for the newly read tag.

Syntax

```
function OnRFIDReadNewTag ( tagObject )
```

tagObject – handle to RFIDTag object describing the new tag.

Example

This example is from the RFIDInventory sample. It does not use the new tag information but simply updates a total count to inform the user.

```
/* OnRFIDReadNewTag */
function OnRFIDReadNewTag( tag )
{
    Message.Text = "Inventory Active\rTag count: " +
                       Reader.TotalTagCount;
}
```

4.4 THE ONRFIDFILTERMATCH EVENT

The `OnRFIDFilterMatch` event is fired when a newly read tag matches a pass or drop filter. The event handler receives a handle to a tag object containing the data for the tag. Use the TagFilterDropSound or TagFilterPassSound properties for a simple tone notification of the filter action. Any script should be brief to avoid slowing the inventory operation.

Syntax

```
function OnRFIDFilterMatch ( filterIndex, tagObject )
```

filterIndex – index specified when filter was defined.
tagObject – handle to RFIDTag object describing the new tag.

Example

This example shows how to keep a count of the filter matches.

```
/* OnRFIDFilterMatch */

var matchCounts = [0, 0, 0, 0];

function OnRFIDFilterMatch ( filterIndex, tagObject )
{
    // Increment count.
    ++matchCounts[filterIndex];
}
```

Appendix 1 – RFID Constants

This appendix contains various constants that are used by CETerm RFID Objects. Many of these constants are a direct representation of the equivalent values from the Symbol RFID APIs and constants. These constants are presented as JavaScript variables for direct inclusion in scripts.

GENERAL CONSTANTS

```
// RFID Utils Constants
// Contains constants and utility functions for Symbol RFID

// Constants are taken from the Symbol rfiddefs.h file.
// * $Revision: 1.8 $
// * $Date: 2006/02/15 23:15:26 $
// * $Author: yjiang $

// Tag types
var RFID_TAG_TYPE_OTHER             = 0;
var RFID_TAG_TYPE_EPC_CLASS0        = 1;
var RFID_TAG_TYPE_EPC_CLASS0PLUS    = 2;
var RFID_TAG_TYPE_EPC_CLASS1        = 3;
var RFID_TAG_TYPE_EPC_CLASSG2       = 4;
var RFID_TAG_TYPE_EPC_CLASS0ZUMA    = 5;

// Filter constants
var RFID_FILTER_OPTION_PASS         = 0;
var RFID_FILTER_OPTION_DROP         = 1;

// CETerm special filter options
var RFID_FILTER_OPTION_PLAYSOUND        = 0x80000000;
var RFID_FILTER_OPTION_FIRESCRIPTEVENT  = 0x40000000;
```

GEN2 CONSTANTS

```
// Memory bank options for Gen 2 write tags memory bank command (21h)
var RFID_GEN2_MEMORY_BANK_RESERVED  = 0;
var RFID_GEN2_MEMORY_BANK_EPC       = 0x01;
var RFID_GEN2_MEMORY_BANK_TID       = 0x02;
var RFID_GEN2_MEMORY_BANK_USER      = 0x03;

// options for for RFID_READCAP_G2_SEL
var RFID_GEN2_SEL_IGNORE_SL         = 0;
var RFID_GEN2_SEL_SL_NOT_SET        = 2;
var RFID_GEN2_SEL_SL_SET            = 3;

// options for RFID_READCAP_G2_SESSION
var RFID_GEN2_SESSION_S0            = 0;
var RFID_GEN2_SESSION_S1            = 1;
```

```
var RFID_GEN2_SESSION_S2            = 2;
var RFID_GEN2_SESSION_S3            = 3;

// options for RFID_READCAP_G2_TARGET
var RFID_GEN2_INVENTORIED_FLAG_A    = 0;
var RFID_GEN2_INVENTORIED_FLAG_B    = 1;


// writing options for Gen 2 write tags' memory bank
var RFID_GEN2_WRITE_OPTION_NONE                             = 0;
var RFID_GEN2_WRITE_OPTION_BLOCK_WRITE                      = 0x01;
var RFID_GEN2_WRITE_OPTION_BLOCK_ERASE_FIRST                = (0x01<<1);
var RFID_GEN2_WRITE_OPTION_PROGRAM_EPC                      = (0x01<<2);

// Lock options for Gen 2 write tags' memory bank
var RFID_GEN2_LOCK_OPTION_NONE                              = 0;
var RFID_GEN2_LOCK_OPTION_PERMA_LOCK_AFTER_WRITE            = 0x01;
var RFID_GEN2_LOCK_OPTION_USE_PERMA_LOCK_BIT                = (0x01<<1);
var RFID_GEN2_LOCK_OPTION_PWD_WRITE_AFTER_WRITE             = (0x01<<2);
var RFID_GEN2_LOCK_OPTION_USE_PWD_WRITE_BIT                 = (0x01<<3);
var RFID_GEN2_LOCK_OPTION_PERMA_LOCK_BANK2_AFTER_WRITE      = (0x01<<4);
var RFID_GEN2_LOCK_OPTION_USE_PERMA_LOCK_BIT2               = (0x01<<5);
var RFID_GEN2_LOCK_OPTION_PWD_WRITE_BANK2_AFTER_WRITE       = (0x01<<6);
var RFID_GEN2_LOCK_OPTION_USE_PWD_WRITE_BIT2                = (0x01<<7);

// Lock mask
var RFID_GEN2_LOCK_MASK_NONE                                = 0;
var RFID_GEN2_LOCK_MASK_USER_PERM                           = (0x01<<0);
var RFID_GEN2_LOCK_MASK_USER_PWD                            = (0x01<<1);
var RFID_GEN2_LOCK_MASK_TID_PERM                            = (0x01<<2);
var RFID_GEN2_LOCK_MASK_TID_PWD                             = (0x01<<3);
var RFID_GEN2_LOCK_MASK_EPC_PERM                            = (0x01<<4);
var RFID_GEN2_LOCK_MASK_EPC_PWD                             = (0x01<<5);
var RFID_GEN2_LOCK_MASK_ACCESSPWD_PERM                      = (0x01<<6);
var RFID_GEN2_LOCK_MASK_ACCESSPWD_PWD                       = (0x01<<7);
var RFID_GEN2_LOCK_MASK_KILLPWD_PERM                        = (0x01<<8);
var RFID_GEN2_LOCK_MASK_KILLPWD_PWD                         = (0x01<<9);

// Lock action
var RFID_GEN2_LOCK_ACTION_NONE                              = 0;
var RFID_GEN2_LOCK_ACTION_USER_PERM                         = (0x01<<0);
var RFID_GEN2_LOCK_ACTION_USER_PWD                          = (0x01<<1);
var RFID_GEN2_LOCK_ACTION_TID_PERM                          = (0x01<<2);
var RFID_GEN2_LOCK_ACTION_TID_PWD                           = (0x01<<3);
var RFID_GEN2_LOCK_ACTION_EPC_PERM                          = (0x01<<4);
var RFID_GEN2_LOCK_ACTION_EPC_PWD                           = (0x01<<5);
var RFID_GEN2_LOCK_ACTION_ACCESSPWD_PERM                    = (0x01<<6);
var RFID_GEN2_LOCK_ACTION_ACCESSPWD_PWD                     = (0x01<<7);
var RFID_GEN2_LOCK_ACTION_KILLPWD_PERM                      = (0x01<<8);
var RFID_GEN2_LOCK_ACTION_KILLPWD_PWD                       = (0x01<<9);
```

READCAP AND WRITECAP CONSTANTS

```
// Selected raw capability access

var RFID_READCAP_RF_ATTENUATION             = 0x0001;

var RFID_WRITECAP_RF_ATTENUATION            = 0x0004;
var RFID_WRITECAP_TAGTYPE                   = 0x0005;

var RFID_TAGCAP_LOCKCODE                    = 0x0006;

var RFID_DEVCAP_IP_PORT                     = 0x000a;
var RFID_TAGCAP_C0_SINGULATION_FIELD        = 0x000b;

var RFID_READCAP_METHOD                     = 0x000d;
var RFID_READCAP_OUTLOOP                    = 0x000e;
var RFID_READCAP_INLOOP                     = 0x000f;

var RFID_DEVCAP_ANTENNA_SEQUENCE            = 0x0010;

var RFID_READCAP_READMODE                   = 0x0011;

var RFID_WRITECAP_ANTENNA                   = 0x0012;

var RFID_WRITECAP_C0_PAGE                   = 0x0013;
var RFID_READCAP_C0_PAGE                    = 0x0014;

var RFID_READCAP_DUTYCYCLE_ONTIME           = 0x0016;
var RFID_READCAP_DUTYCYCLE_OFFTIME_PCNT     = 0x0017;

// Gen 2 inventory
var RFID_READCAP_G2_SEL                     = 0x0018;
var RFID_READCAP_G2_SESSION                 = 0x0019;
var RFID_READCAP_G2_TARGET                  = 0x001A;
var RFID_READCAP_G2_START_Q                 = 0x001B;

// Gen 2 read tag's memory bank (20h)
var RFID_TAGCAP_G2_SELECT_MEM_BANK          = 0x001C;
var RFID_TAGCAP_G2_MEM_BANK                 = 0x001D;
var RFID_TAGCAP_G2_WORD_POINTER             = 0x001E;

var RFID_WRITECAP_G2_WRITE_OPTIONS          = 0x001F;
var RFID_WRITECAP_G2_LOCK_OPTIONS           = 0x0020;

// Gen 2 lock tags' memory (23h)
var RFID_WRITECAP_G2_LOCK_MASK              = 0x0021;
var RFID_WRITECAP_G2_LOCK_ACTION            = 0x0022;

// Gen 2 kill tag (24h)
var RFID_TAGCAP_G2_ACCESS_PASSWORD          = 0x0023;
var RFID_TAGCAP_G2_KILL_PASSWORD            = 0x0024;
```

TRIGGER CONSTANTS

```
// Trigger status values
var TRIGGER_STAGE1_MASK   = 0x000000ff;
var TRIG1_STAGE1          = 0x00000001;
var TRIG2_STAGE1          = 0x00000002;
var TRIG3_STAGE1          = 0x00000004;
var TRIG4_STAGE1          = 0x00000008;
var TRIG5_STAGE1          = 0x00000010;
var TRIG6_STAGE1          = 0x00000020;
var TRIG7_STAGE1          = 0x00000040;
var EXT_TRIG_STAGE1       = 0x00000080;

var TRIGGER_STAGE2_MASK   = 0x0000ff00;
var TRIG1_STAGE2          = 0x00000100;
var TRIG2_STAGE2          = 0x00000200;
var TRIG3_STAGE2          = 0x00000400;
var TRIG4_STAGE2          = 0x00000800;
var TRIG5_STAGE2          = 0x00001000;
var TRIG6_STAGE2          = 0x00002000;
var TRIG7_STAGE2          = 0x00004000;
var EXT_TRIG_STAGE2       = 0x00008000;

var TRIGGER_ALL_MASK      = ( TRIGGER_STAGE1_MASK |
TRIGGER_STAGE2_MASK );
```

UTILITY FUNCTIONS

```
// Convert hex digits in form "10 20 30" to bit mask representation.
// Hex digits must contain the offset (skipped) locations
// Allow comma and "0x" prefix on each hex digit pair.

function BitMaskFromHex( startOffset, bitCount, hexString )
{
    var bitMask = "";

    // Split hexString into array
    var hexArray = hexString.split( /[,\s]+/ );

    // Build skipped locations
    for (var j=0; j<startOffset; ++j)
    {
        bitMask += "x";
    }

    var bitIndex = startOffset;
    var byteHexValue;
    for (var k=startOffset; k<bitCount; ++k)
    {
        // Verify that there are enough hex digits
        if (bitIndex/8 >= hexArray.length)
```

```
            {
                return null;
            }

            byteHexValue = hexArray[Math.floor(bitIndex / 8)];
            if (!byteHexValue.match( /0x/i ))
            {
                // Prepend hex designator if missing
                byteHexValue = "0x" + byteHexValue;
            }
            bitMask += (byteHexValue & (0x1 << (7-(bitIndex % 8))) ) ?
                                                    '1' : '0';
            ++bitIndex;
        }

        return bitMask;
    }
```

Appendix 2 – RFID Inventory Sample Scripts

This appendix contains the RFID Inventory Sample application scripts.

```
// RFID Utils Constants
// Symbol RFID Inventory Sample

var RFIDTriggerAllowed = false;
var RFIDInventoryActive = false;
var Message = CETerm.Message;
var Reader = CETerm.RFIDReader;

/* OnRFIDInventoryDone */
function OnRFIDInventoryDone( status, totalTags, newTags )
{
    // Check our global flag for active inventory
    if (RFIDInventoryActive)
    {
        // Start another inventory cycle
        Reader.StartTagInventory();
    }
    else
    {
        // Process inventory data
        Message.ProgressRunning = false;
        Message.Progress = 0;

        if (CETerm.Session(CETerm.ActiveSession).Browser.DocLoaded)
        {
            // Hide information message
            Message.IsVisible = false;

            // Build table to return data.
            var tag;
            var tagTable = "'";
            tagTable +=
                    '<font size=1>' +
                    '<table width="100%" bgcolor="tan">' +
                    '<tr>' +
                    '<td align="center" width="5%">#</td>' +
                    '<td align="center" width="5%">Cls</td>' +
                    '<td align="center" width="5%">#Rd</td>' +
                    '<td align="center" width="85%">ID</td></tr>';

            for (var j=0; j<totalTags; ++j)
            {
                tag = Reader.GetTag(j);
                tagTable += "<tr><td>" + (j+1) + "</td><td>";
                switch (tag.Type) {
                case RFID_TAG_TYPE_EPC_CLASS0:
                    tagTable += "C0";
                    break;
                case RFID_TAG_TYPE_EPC_CLASS1:
                    tagTable += "C1";
                    break;
                case RFID_TAG_TYPE_EPC_CLASSG2:
```

```
                    tagTable += "G2";
                    break;
                default:
                    tagTable += "?";
                }
                tagTable += "</td><td>" + tag.ReadCount +
                            "</td><td>" + tag.ID + "</td></tr>";

            }
            tagTable += "</table></font>'";

            var script = "OnRFIDInventoryDone("+ status + "," +
                                totalTags + "," +
                                Reader.TotalReadCount + "," +
                                tagTable + ");"
            var b = CETerm.Session(CETerm.ActiveSession).Browser;
            b.RunScript( script );
        }
        else
        {
            // TE session
            // Show success message
            Message.Text = "Inventory Done\rtotalTags=" + totalTags +
                      " newTags=" + newTags +
                      " status=" + status +
                      "\rPull Trigger to continue...\r";

            // Send tag data to current session
            for (var j=0; j<totalTags; ++j)
            {
                CETerm.SendText( "#" + Reader.GetTagID(j), 0);
            }

            // Message will vanish in 2 seconds
            Message.Timeout = 2;
        }

        // Prepare for next read
        Reader.ClearInventory();
        RFIDTriggerAllowed = true;
    }
}



/* OnRFIDTrigger */
function OnRFIDTrigger( status )
{
    if (RFIDTriggerAllowed &&
        (status & TRIGGER_ALL_MASK))
    {
        // Trigger pulled
        RFIDTriggerAllowed = false;
```

```
            RFIDInventoryActive = true;
            Message.Text = "Inventory Active\rTag count: 0";
            Message.Timeout = 0;
            Message.IsVisible = true;
            Message.ProgressRunning = true;

            Reader.ReaderPowerOn = true;
            Reader.StartTagInventory();
        }
        else
        {
            // If trigger not enabled or released
            // Stop inventory
            RFIDInventoryActive = false;

            // Usually PowerOn = false will stop inventory.
            // Sometimes if the trigger is released too quickly it
            // fails to stop the inventory.
            // To recover, pull and release trigger
            Reader.ReaderPowerOn = false;
        }
    }


    /* OnRFIDReadNewTag */
    function OnRFIDReadNewTag( tag )
    {
        Message.Text = "Inventory Active\rTag count: " +
                                Reader.TotalTagCount;
    }


    function RFIDUpdateBrowserParams()
    {
        var doc = CETerm.Session(CETerm.ActiveSession).Browser.Document;

        // Stop any current reader action
        Reader.ReaderPowerOn = false;

        Reader.TagClass1Enabled = doc.form1.Class1.checked;
        Reader.TagClass0Enabled = doc.form1.Class0.checked;
        Reader.TagClassGen2Enabled = doc.form1.Gen2.checked;

        Reader.NewTagSoundEnabled = doc.form1.NewTagBeep.checked;

        if (doc.form1.SelectDevice[0].checked)
        {
            Reader.EnableTrigger( true );
            Reader.EnableReader( true );
        }
        else
        {
            Reader.EnableTrigger( false );
```

```
            Reader.EnableReader( false );
        }

    }



    /* RFIDReadOneTag */
    function RFIDReadOneTag()
    {
        var result = Reader.ReadTag();

        OS.Alert( "tag=" + (result ?
                        Reader.GetCommandStatusText( result ) :
                        Reader.GetTagID(0) ));
    }



    function RFIDInitInventory()
    {
        RFIDTriggerAllowed = true;
        Reader.EnableTrigger( true );
        Reader.EnableReader( true );
        Reader.ClearInventory();

        Reader.TagClass1Enabled = true;
        Reader.TagClass0Enabled = true;
        Reader.TagClassGen2Enabled = true;

        Reader.NewTagSoundEnabled = true;
        Reader.NewTagScriptEnabled = true;
        Reader.NewTagSound = "nledtemp.wav";

        // Filters
        Reader.TagFilterRemoveAll();

        // Display message box
        Message.Title = "RFID Tag Inventory";
        Message.Text = "Waiting for trigger...";
        Message.Timeout = 0;
        Message.AbortButtonVisible = false;
        Message.ProgressVisible = true;
        Message.ProgressRunning = false;
        Message.IsVisible = false;
    }
```

Appendix 3 – RFID Inventory Sample HTML

This appendix contains the RFID Inventory Sample HTML page.

```
<html>
<head>
<title>RFID Inventory</title>
<meta http-equiv="TextSize" content="Smallest">
<meta http-equiv="Scanner" content="Enabled">
<meta http-equiv="ScannerNavigate"
content="Javascript:OnScan('%s','%s','%s','%s','%s');">

<meta http-equiv="OnKey_RETURN" content="Javascript:OnReturn();">
<meta http-equiv="OnKey_ESCAPE" content="Javascript:OnEscape();">

<script language=javascript>
// Global variables
var oCETerm = null;
var currentstate = 'reset';
</script>

</head>
<body scroll=no onload="Javascript:LoadActions();">
<form name=form1>
<div id="TopContent">
Error
</div>
<hr align="center" width="95%">
<center>Tag Inventory</center>
<div id="BottomContent">
Error
</div>
<br>
<font size=2>
<hr align="center" width="95%">
Total Tags:
<input type="text" name="TotalTags" value="0" size=4 disabled>
Reads:
<input type="text" name="TotalReads" value="0" size=4 disabled>
<br>
<hr align="center" width="95%">
<input type="checkbox" name="Gen2" value="Gen2"
onclick="UpdateParams();" checked>Gen2
<input type="checkbox" name="Class0" value="Class0"
onclick="UpdateParams();" checked>Class0
<input type="checkbox" name="Class1" value="Class1"
onclick="UpdateParams();" checked>Class1
<br>
<input type="checkbox" name="NewTagBeep" value="NewTagBeep"
onclick="UpdateParams();" checked>New Tag Beep
<input type="radio" name="SelectDevice" value="RFID"
onclick="UpdateParams();" checked>RFID
<input type="radio" name="SelectDevice" value="Barcode"
onclick="UpdateParams();">Barcode
</font>
</form>
```

```
<script language=javascript>

function LoadActions()
{
    // Prepare object
    oCETerm = new ActiveXObject( "Cebrowsex.IdaCtl" );

    // Initialize RFID
    oCETerm.CETerm.RunScript( "RFIDInitInventory();" );

    // Set current display
    SetPageContent();

    // Update params
    UpdateParams();
}


// OnRFIDInventoryDone is called by CETerm script engine
function OnRFIDInventoryDone( status, totalTags,
                              readCount, tagtable )
{
    document.form1.TotalTags.value = totalTags;
    document.form1.TotalReads.value = readCount;
    BottomContent.innerHTML = tagtable;

}


// UpdateParams
function UpdateParams()
{
    // Update RFID params
    oCETerm.CETerm.RunScript( "RFIDUpdateBrowserParams();" );
}


function OnReturn()
{
    SetPageContent();
}


function OnEscape()
{
    // Clear all
}


function SetPageContent()
{
    switch( currentstate )
```

```
    {
    case 'reset':
        // Tag page
        TopContent.innerHTML =
            '<font size=+2><center>Pull trigger' +
            ' to start inventory...<br></center></font>';
        BottomContent.innerHTML =
            '<font size=-1>' +
            '<table width="100%" bgcolor="tan"><tr>' +
                '<td align="center" width="5%">#</td>' +
                '<td align="center" width="5%">Cls</td>' +
                '<td align="center" width="5%">#Rd</td>' +
                '<td align="center" width="85%">ID</td></tr>' +
            '</table>' +
            '</font>';

        break;

    case 'configReader':
        break;
    }
}


function poweron()
{
    alert("Received PowerOn notification.");
}

function OnScan(data, source, type, time, length)
{
  alert("Scanned barcode was " + data +
        "\nThe symbology was " + type +
        "\nScanned at " + time +
        "\nWith a length of " + length);
}

</script>
</body>
</html>
```

Index

C

CETerm object · 10
ClearInventory · 11, 13

E

EnableReader · 11
EnableTrigger · 11
EraseTag · 11, 12

G

GetCapInfo · 11, 12
GetCapList · 11, 12
GetCapValue · 11, 12
GetCommandStatusText · 11, 12
GetStats · 11, 12
GetTag · 11, 13
GetTagID · 11, 13

K

KillTag · 11

L

LockTag · 11

O

OnRFIDFilterMatch · 19, 21
OnRFIDInventoryDone · 19
OnRFIDReadNewTag · 19, 20
OnRFIDTrigger · 19, 20
OS object · 10

P

ProgramTags · 11, 12

R

ReadTag · 11, 12
RFID Constants · 22
RFIDReader object · 10, 14
RFIDTag · 18

S

SetCapValue · 11, 12
SetScrollInventoryDefaults · 11, 12
SetTreeInventoryDefaults · 11, 13
StartTagInventory · 11, 12

T

TagFilterAdd · 11, 13
TagFilterRemove · 11, 13
TagFilterRemoveAll · 11, 13
TextInput · 10